Query Documentation & Code Snippets

# Query Documentation

## Query 1

-- This query creates and displays a view containing the contact information of all female candidates.

-- This could be used by the insurance company to monitor the female population of their users, using the data for International Women’s Day.

-- This query exhibits the WHERE clause.

CREATE VIEW ContactInformationOfFemaleCandidates

AS -- name the view to be created

SELECT -- select information about the candidate

CONCAT(c.CandidateTitle,' ',c.CandidateForename,' ',c.CandidateSurname,' ') AS Full\_Name, -- conjoin atomised names into one field

c.CandidatePhoneNumber AS Phone\_Number,

c.CandidateEmail AS Email,

c.CandidateAddress1 AS Address\_Line\_One,

c.CandidateAddress2 AS Address\_Line\_Two,

c.CandidatePostCode AS Post\_Code

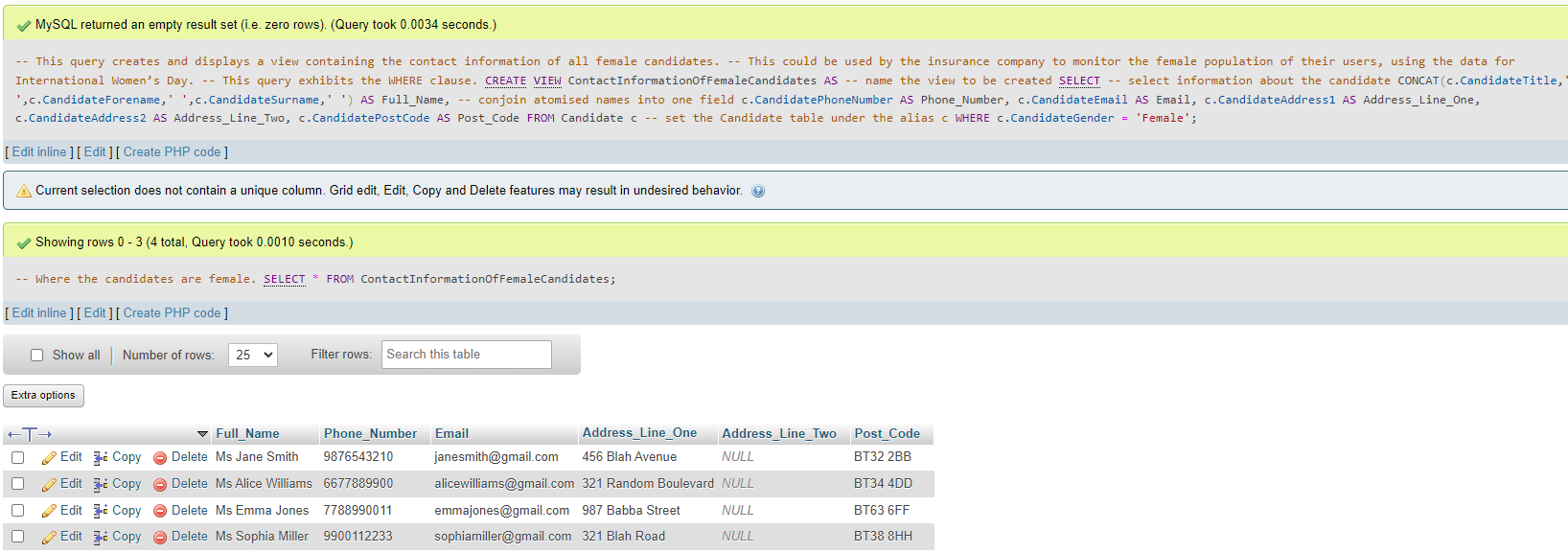
FROM

Candidate c -- set the Candidate table under the alias c

WHERE

c.CandidateGender = 'Female'; -- Where the candidates are female.

SELECT \* FROM ContactInformationOfFemaleCandidates; -- Display this view.



## Query 2

-- This query creates and displays a view containing average wages of every entry in the Business table.

-- This could be displayed to all candidates to show the average wage they should be expecting for each business.

-- This query exhibits Aggregate Functions, Group By, Sort By and Joins.

CREATE VIEW AvgWageFromBusinesses AS -- name the view to be created

SELECT -- select the name of the business alongside the average wage in pounds

b.BusinessName AS Business\_Name,

ROUND(AVG(jb.JobBusinessSalary),2) AS Average\_Wage -- Leave only two decimal points

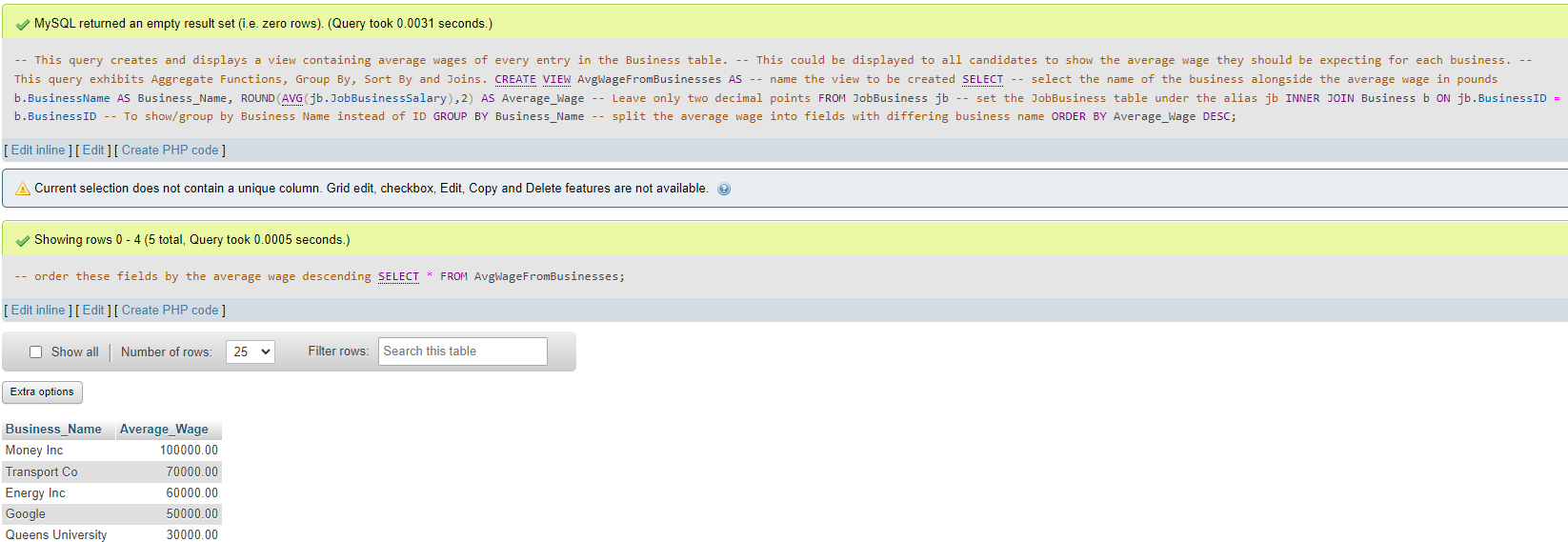
FROM JobBusiness jb -- set the JobBusiness table under the alias jb

INNER JOIN Business b ON jb.BusinessID = b.BusinessID -- To show/group by Business Name instead of ID

GROUP BY Business\_Name -- split the average wage into fields with differing business name

ORDER BY Average\_Wage DESC; -- order these fields by the average wage descending

SELECT \* FROM AvgWageFromBusinesses; -- Display this view



## Query 3

-- This query creates and displays a view containing all the relevant data about an interview

-- This could be displayed to both the candidate and the recruiter before and during the interview

-- This query exhibits three Joins.

CREATE VIEW InterviewInfo AS -- name the view to be created

SELECT

CONCAT(c.CandidateForename,' ',c.CandidateSurname) AS Candidate\_Name, -- conjoin atomic names into one field

i.InterviewRoom AS Interview\_Room,

i.InterviewDate AS Interview\_Date,

i.InterviewTime AS Interview\_Time,

b.BusinessName AS Business\_Name,

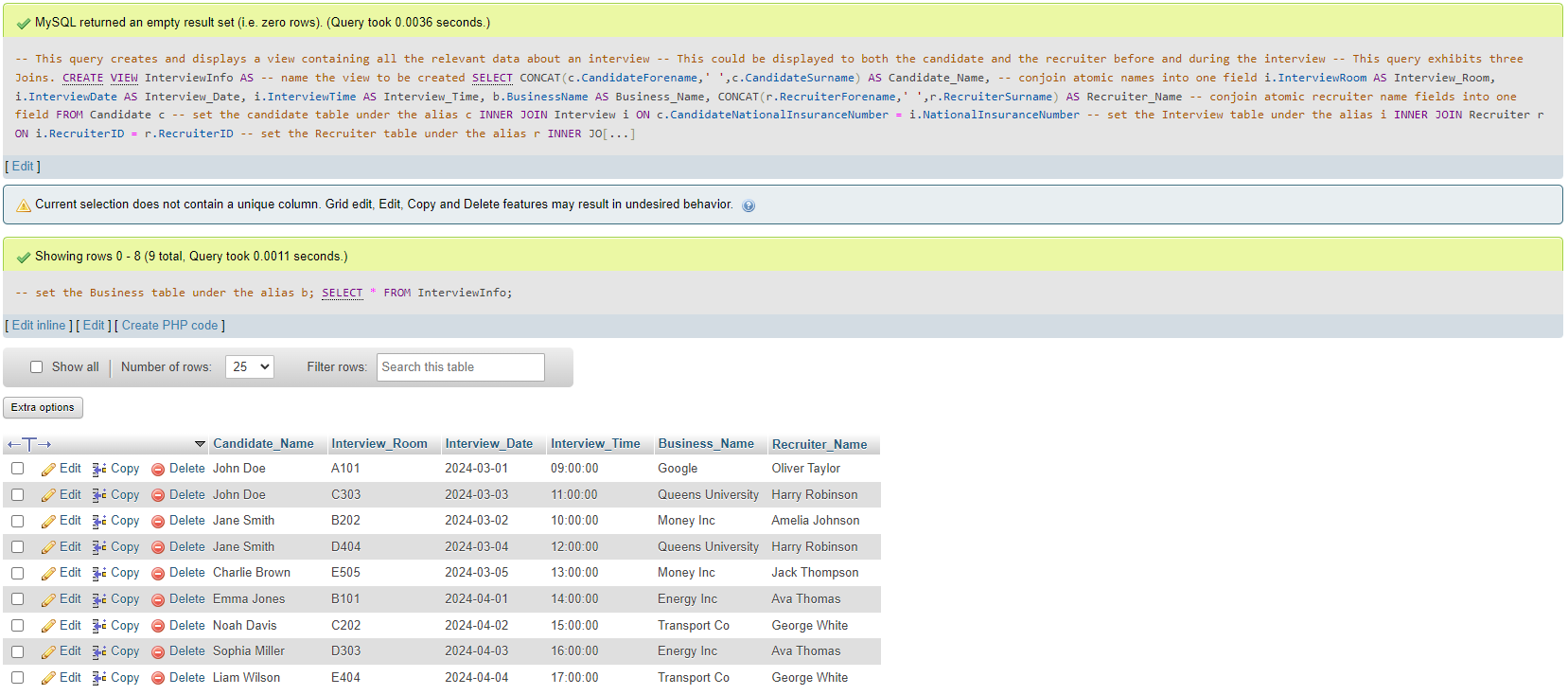
CONCAT(r.RecruiterForename,' ',r.RecruiterSurname) AS Recruiter\_Name -- conjoin atomic recruiter name fields into one field

FROM Candidate c -- set the candidate table under the alias c

INNER JOIN Interview i ON c.CandidateNationalInsuranceNumber = i.NationalInsuranceNumber -- set the Interview table under the alias i

INNER JOIN Recruiter r ON i.RecruiterID = r.RecruiterID -- set the Recruiter table under the alias r

INNER JOIN Business b ON r.BusinessID = b.BusinessID; -- set the Business table under the alias b;

SELECT \* FROM InterviewInfo; -- Display this view

## Query 4

-- This query creates and displays a view containing all the relevant data about all interviews in March 2024 with a recruiter from the company Google

-- This could be used by Google to monitor their rate of interviewing and the performance of their Recruiters

-- This query exhibits three Joins, ordering and filtering records within a specific date range.

CREATE VIEW InterviewsInCompanyTimeframe AS -- name the view to be created

SELECT

CONCAT(c.CandidateForename,' ',c.CandidateSurname) AS Candidate\_Name,-- conjoin atomic names into one field

i.InterviewDate AS Interview\_Date,

i.InterviewTime AS Interview\_Time,

CONCAT(r.RecruiterForename, ' ', r.RecruiterSurname) AS Recruiter\_Name, -- conjoin atomic reviewer names into one field

b.BusinessName as Business\_Name

FROM Interview i -- set the Interview table under the alias i

JOIN Candidate c ON i.NationalInsuranceNumber = c.CandidateNationalInsuranceNumber -- set the Candidate table under the alias c

JOIN Recruiter r ON i.RecruiterID = r.RecruiterID -- set the Recruiter table under the alias r

JOIN Business b ON r.BusinessID = b.BusinessID -- set the Business table under the alias b

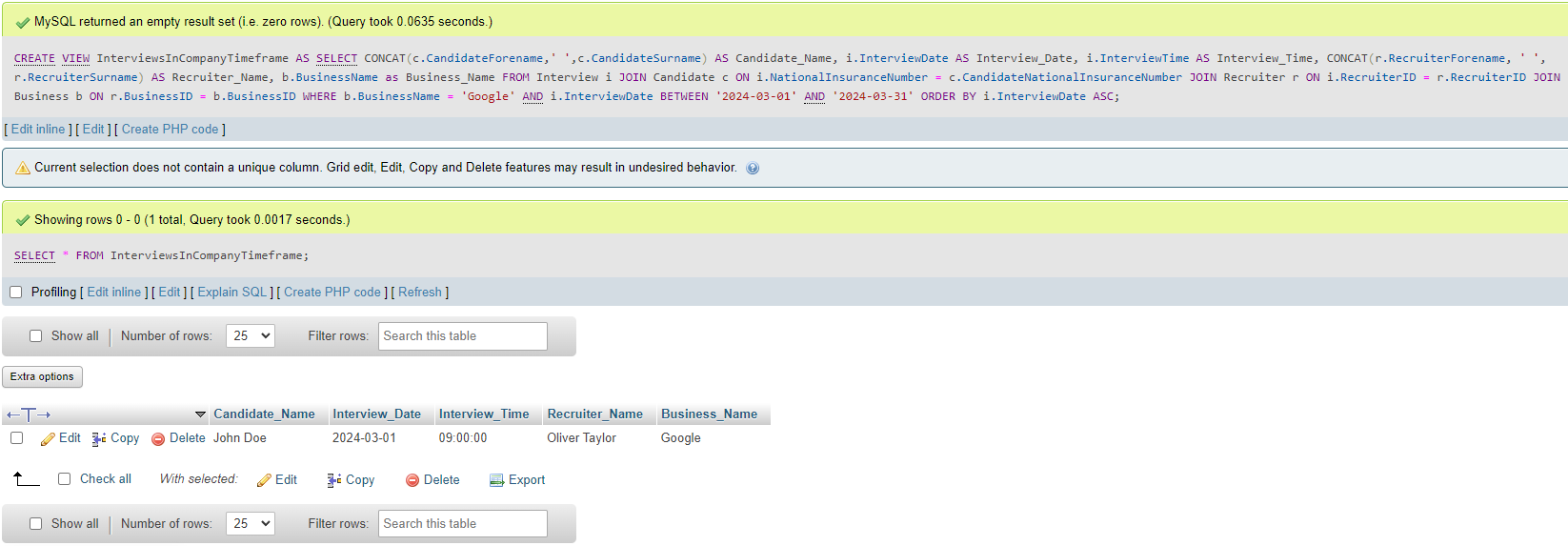
WHERE -- only select rows where the business is Google and the interview date is within the month of March 2024

b.BusinessName = 'Google'

AND

i.InterviewDate BETWEEN '2024-03-01' AND '2024-03-31'

ORDER BY i.InterviewDate ASC; -- order these fields by the average ascending

SELECT \* FROM InterviewsInCompanyTimeframe; -- Display this view

# Code Snippets

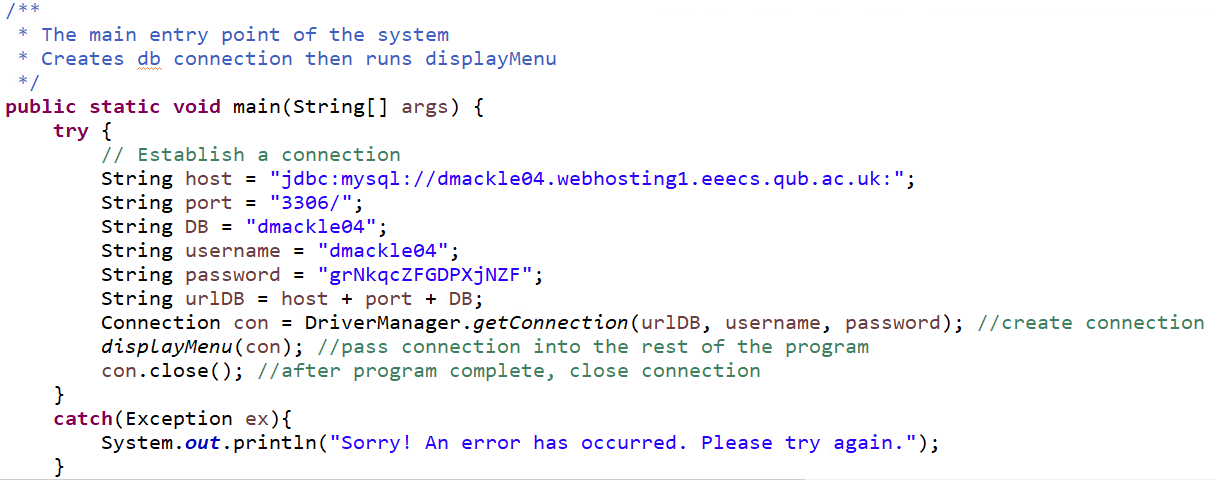
## main

This code snippet shows my ‘main’ method, which is the main entry point of the program.

Using the java.sql library, a connection to the sql server is built and I am authenticated, giving me access to my database.

The displayMenu method is run, and the created sql connection is passed to it as an argument..

After this method completes, the connection is closed and the program ends.



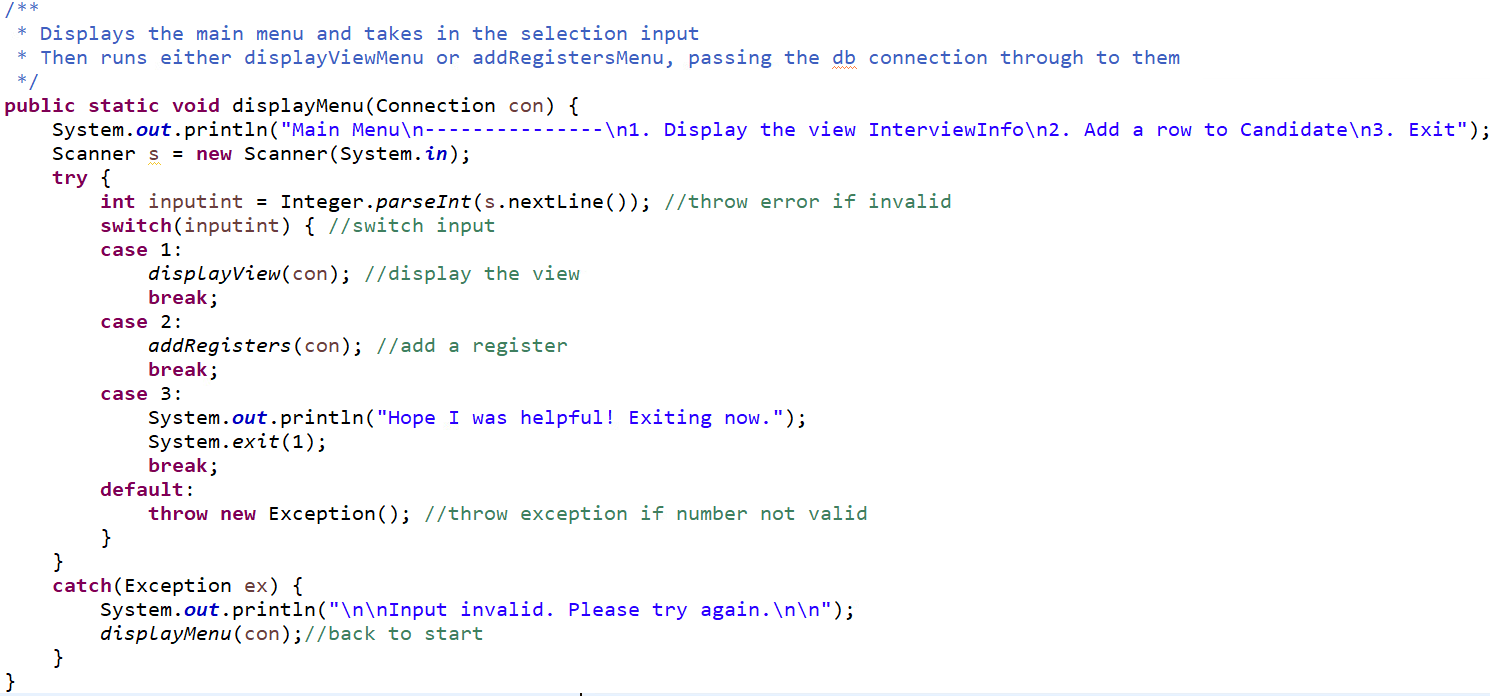
## 

## displayMenu

This code snippet displays my main menu, with three options: Displaying a View, Adding a Register or Exiting. The user is asked for an input, which is used to navigate between these three selections.

Option 1 and 2 run a method which either opens the menu to Display a View or the menu to Add a Register.

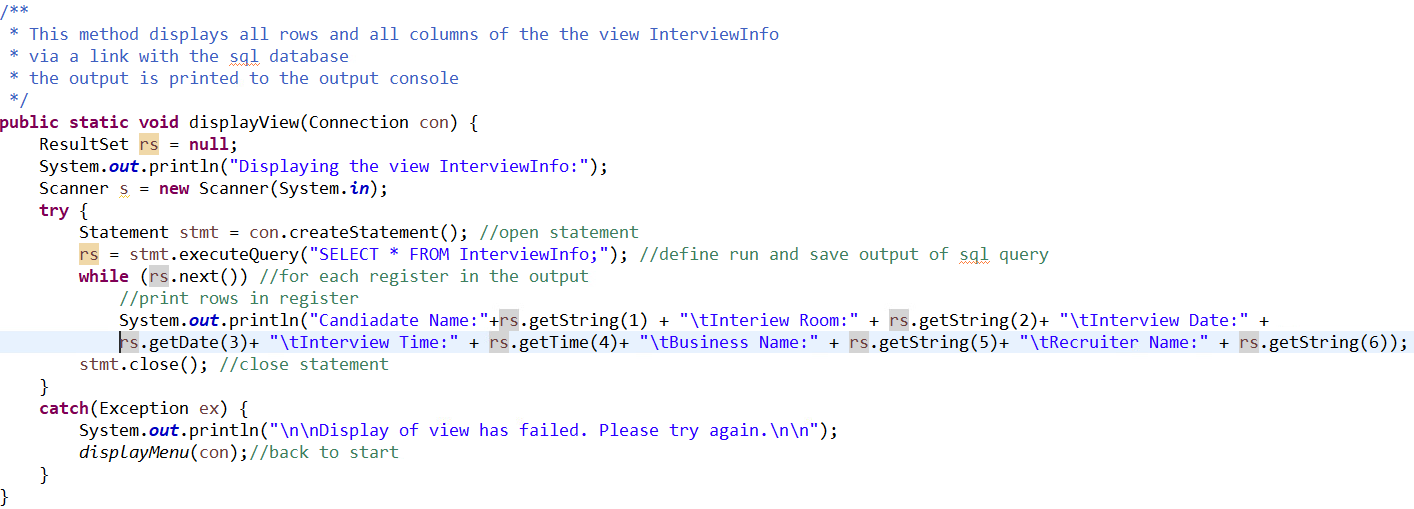
Option 3 exits the program fully and safely.



## displayView

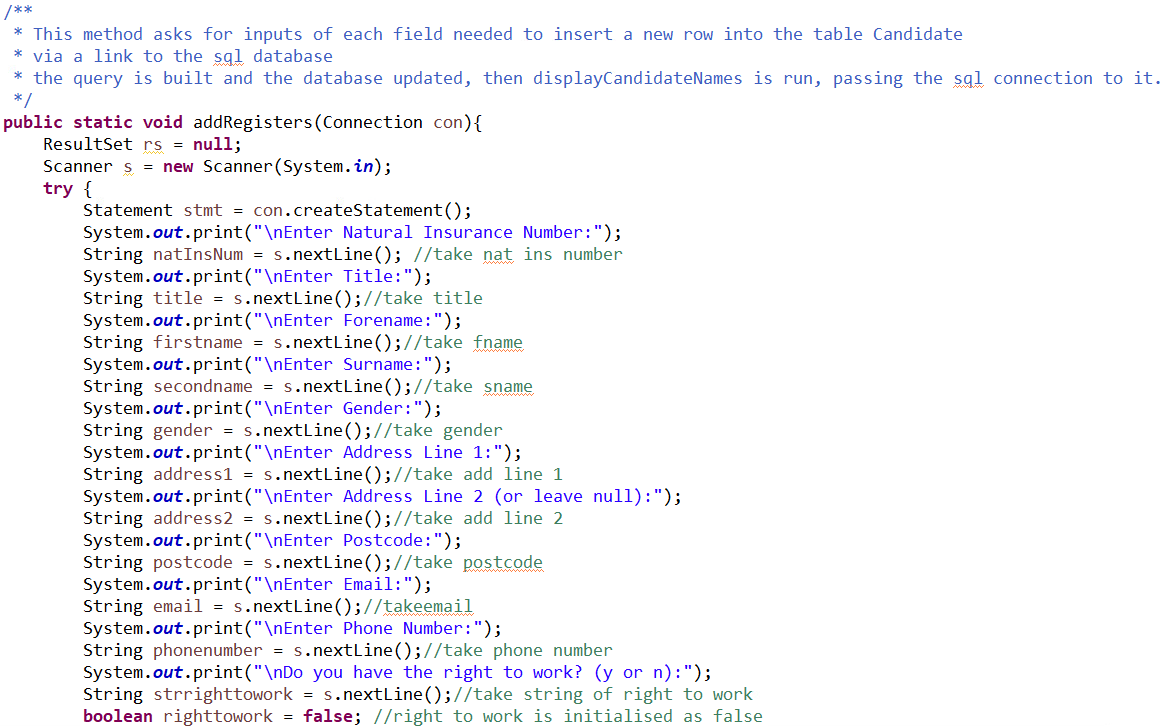
This code snippet displays all rows and columns of the view InterviewInfo.

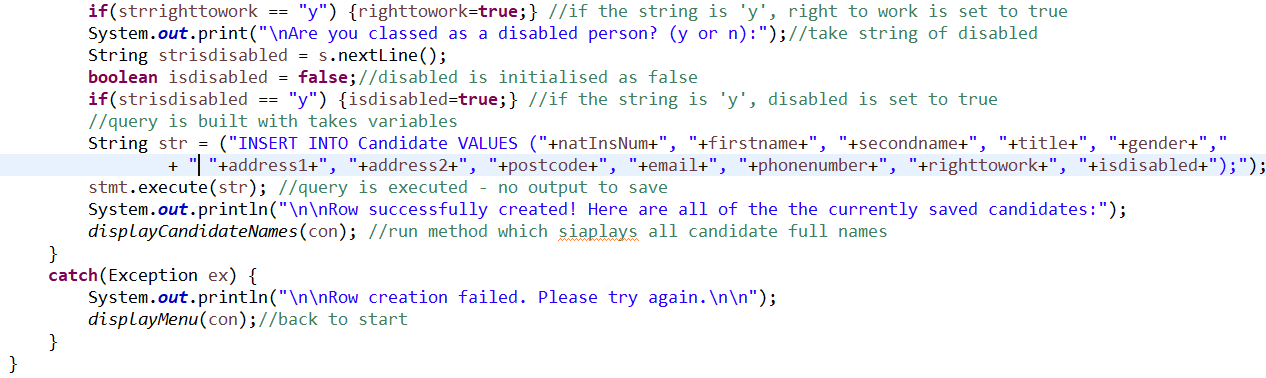
The preset query is run, and for each row of the output, all attributes of that row are displayed onto the console.

This method uses a Statement object, which holds and executes the SQL query. This Statement object is closed after use.

## addRegisters

This code snippet displays the addRegisters method, which takes in inputs for each of the fields to create a SQL query which inserts another row into the table Candidate. Most inputs are SQL VARCHARs, which can be taken as Strings in java, but the SQL BOOLEAN data type must be taken as a String, and then converted into a boolean in java by checking if the string is equal to “y”.

This method uses a Statement object, which holds and executes the SQL query. This Statement object is closed after use.



## displayCandidateNames

This snippet displays the method displayCandidateNames, which displays the full names of all current rows in the Candidate Table.

The full name to be displayed is created from three attributes stored in Candidate, Candidate Title, CandidateForename and CandidateSurname.

The preset SQL Query to output these rows is run, and the output harvested.

For each row of the output, the details are displayed to the java output console.

This method uses a Statement object, which holds and executes the SQL query. This Statement object is closed after use.

